Tópico 03 - Visão geral da função do computador; interconexão do computador

Hugo Vinícius Leão e Silva

hugovlsilva@gmail.com, hugo.vinicius.16@gmail.com, hugovinicius@ifg.edu.br

Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Campus Anápolis
Curso de Bacharelado em Ciência da Computação

31 de maio de 2021

Visão geral

Estrutura e função do computador

- Conhecer a responsabilidade de cada componente (CPU, RAM, E/S) e como eles se interconectam permite:
  - Detectar gargalos **do computador** como um todo;
  - Estimar o tamanho das falhas de **um computador** caso um componente falhe;
  - Propor soluções de maior desempenho e/ou de maior confiabilidade para o **computador como um todo**;
- Conceitos principais da Arquitetura de Von Neumann:
  - Dados e instruções armazenados em uma **única** memória de leitura e escrita...
  - ... que são acessados unicamente pelo **endereço de memória**;
  - A execução do programa obedece ao ciclo de instrução.
  - Por isso, um programa pode ser implementado em *hardware* ligando as portas lógicas e os flip-flops com a estrutura necessária – programa ***hardwired***.

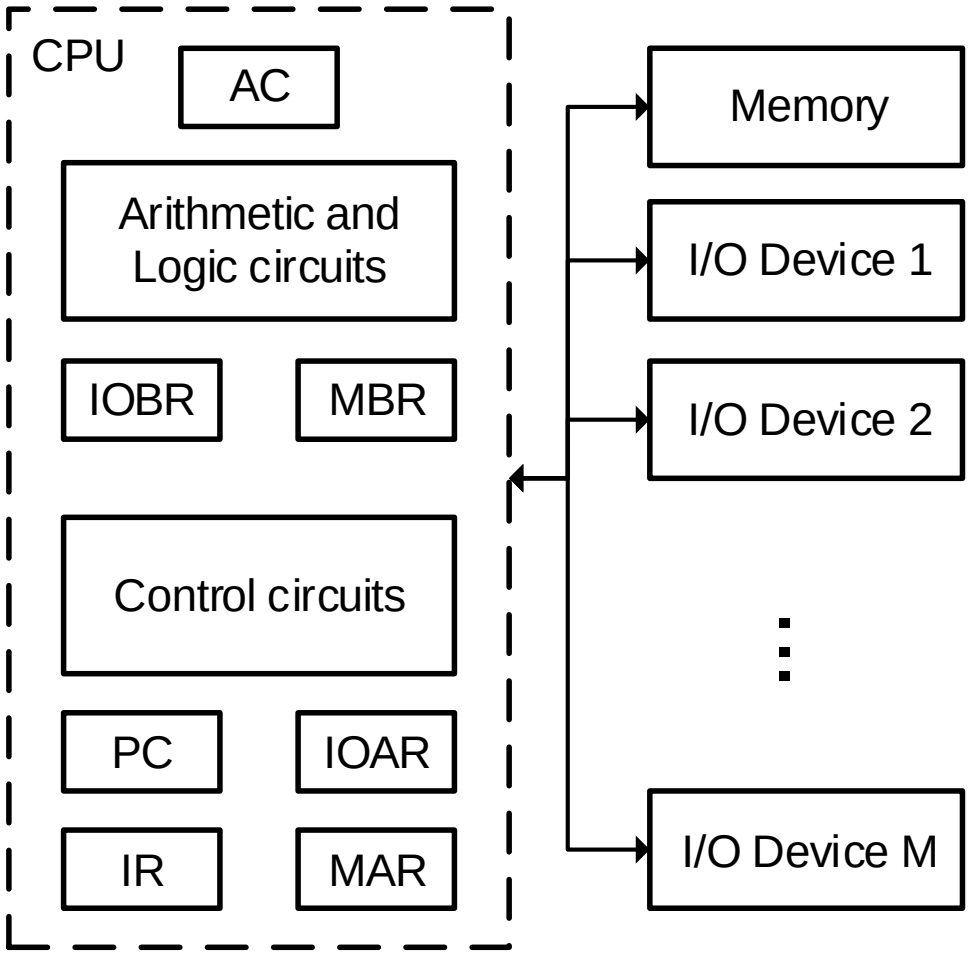
Estrutura e função do computador

- O *hardware* personalizado com o programa *hardwired* opera sobre os dados dependendo dos sinais de controle aplicados a ele;
- Em um *hardware* de propósito geral, o programa está em *software* na forma de instruções;
- As instruções geram os sinais de controle adequados para executar o *software*, sem ter que refazer a estrutura de *hardware*;
- Para isso, é necessário um **estágio de decodificação de instruções** que gera os sinais de controle para ativar e desativar circuitos digitais conforme a necessidade;

Estrutura e função do computador

- Então, é necessário um dispositivo de entrada (no ponto de vista da CPU) que fornece instruções e dados – **unidade de memória** – utilizando o MAR e o MBR como *buffers*;
- A CPU troca dados com os dispositivos de E/S da mesma forma – os dispositivos de E/S são endereçáveis e existem registradores na CPU para E/S como o MAR e o MBR.

Figura: Diagrama básico da CPU, memória e dispositivos de E/S no IAS



Estrutura e função do computador

- Mais uma vez: ciclo de instrução? Busca-Decodifica-Executa!
- Ciclo de busca do IAS:
  - `PC` – *Program Counter* – indica o endereço de memória do próximo par de instruções a ser executado;
  - A não ser que haja saltos (jumps), a próxima instrução sempre é `PC = PC + 1`.
- Ciclo de decodificação do IAS:
  - O *opcode* da instrução que será executada está em `IR` – *Instruction Register*;
  - A Unidade de Controle ativa os sinais de controle corretos para executar a instrução.
- Ciclo de Execução do IAS:
  - A instrução é executada.

Estrutura e função do computador

Há quatro tipos de instrução:

- **Processador-Memória** – troca dados entre CPU e RAM;
- **Processador-E/S** – troca dados entre CPU e E/S;
- **Processamento** – processa dados;
- **Controle** – exemplo: altera a sequência de execução do programa.

Estrutura e função do computador

Considere a instrução: `ADD A, B` $\rightarrow$ `A = A + B`, onde `A` e `B` são posições de memória

Ciclo de instrução do DEC PDP-11:

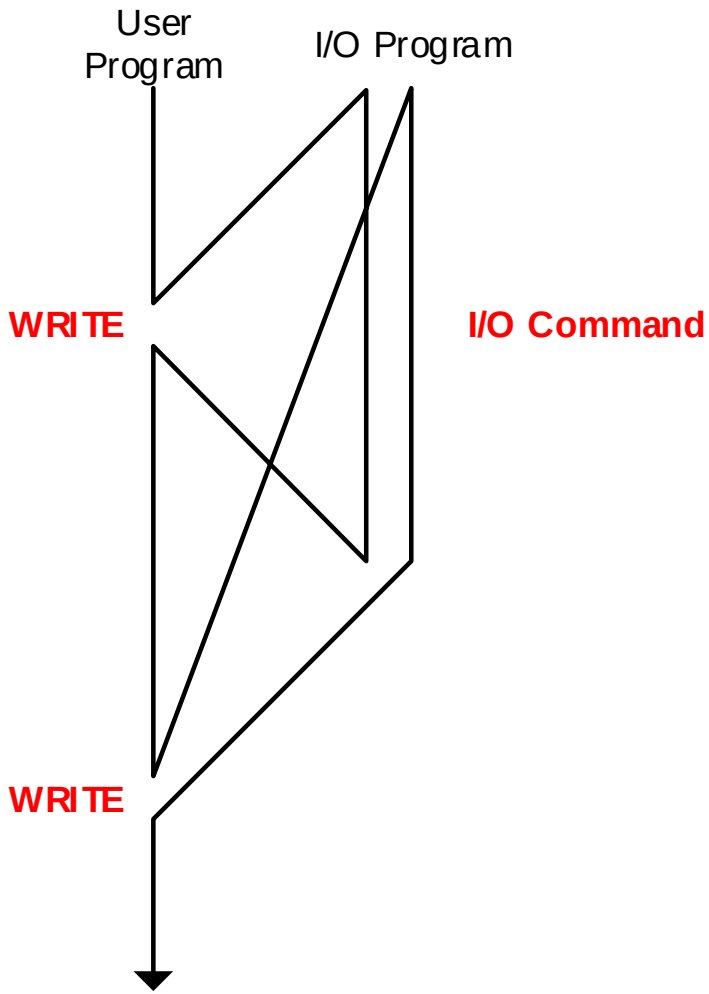
1. **Busca `ADD` na memória**;
2. Decodifica `ADD`;
3. **Busca `A` na memória e armazena em um registrador interno (`r0`)**;
4. **Busca `B` na memória e armazena em outro registrador interno (`r1`)**;
5. `r0 = r0 + r1`;
6. Armazena o conteúdo de `r0` na posição de memória `A`.

Observa-se que ciclo de instrução é mais **complexo** – duas referências à memória;

É necessário calcular o endereço de 1) `ADD`, 2) `A` e 3) `B` antes de buscá-los/armazená-los na memória (AGU – *Address Generation Unit*).

- Todos os computadores possuem **interrupção** para interromper o ciclo de instrução;
- Dispositivos de E/S são **muito** mais lentos que a CPU;
- Exemplo:
  - Um programa envia um documento para ser impresso considerando o ciclo de instrução padrão;
  - Cada vez que a CPU transfere dados para a impressora via memória, ele **deve** esperar;
  - **Enquanto isso, poderia fazer outros processamentos**.

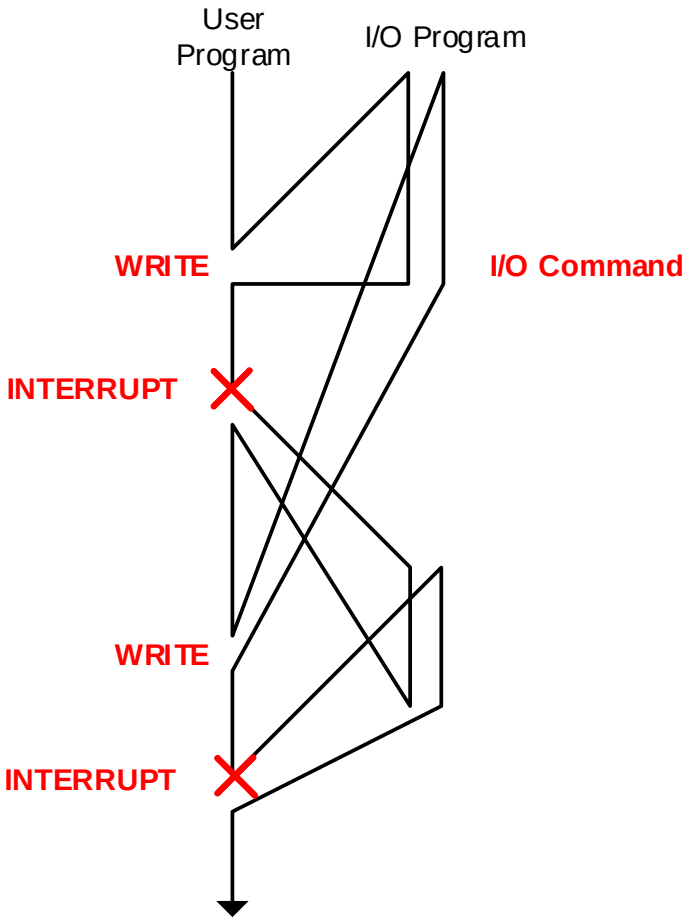
Figura: Fluxo de um programa fazendo chamadas de E/S



Estrutura e função do computador – Interrupções

- Uma solução: **interrupção**;
- Executa-se o comando de E/S (ou operação de E/S);
- Logo após retorna-se à execução do programa de usuário **enquanto** a operação de E/S é processada pelo dispositivo de E/S;
- Quando o dispositivo de E/S está pronto para receber mais dados da CPU, envia um sinal de **Requisição de Interrupção** (IRQ – *Interrupt Request*);
- A CPU suspende a execução do programa atual e executa o programa E/S para atender o dispositivo – tradador de interrupção;
- Isso é feito pela CPU em conjunto com o SO (Sistema Operacional) e é transparente para o programa de usuário.

Figura: Fluxo de um programa fazendo chamadas de E/S com interrupção



- Isso é feito pela CPU em conjunto com o SO (Sistema Operacional) e é transparente para o programa de usuário;
- Ciclo de instrução → **Interrupção**-Busca-Decod.-Executa:
  - Se não houver nenhuma interrupção pendente, segue com o ciclo de instrução normalmente;
  - Se houver, salva o contexto do programa e o suspende. Passa a executar o programa que trata a interrupção.
- Claramente, há um *overhead* – mais instruções precisam ser executadas, mas, ainda assim, a CPU é usada mais eficientemente.

- Um exemplo de interrupções múltiplas: um PC recebendo uma imagem da placa de rede e mostrando na tela;
- Nesse caso, uma solução é: enquanto uma interrupção estiver sendo tratada, a CPU pode desativar as outras interrupções e elas são tratadas **serialmente**;
- Mas aí não leva em consideração **prioridade** ou **tempo crítico**;
- Outra solução é: apenas interrupções com maior prioridade são atendidas enquanto a CPU estiver tratando uma interrupção e elas são tratadas de **maneira aninhada**.

- Em todo caso, o tipo de E/S descrito aqui foi: **E/S Programada** (ou *Programmed I/O*): E/S-**CPU**-RAM;
- Às vezes é desejável, ou mesmo necessário, que o dispositivo de E/S acesse diretamente a memória sem ocupar a CPU – **DMA** (*Direct Memory Access*)

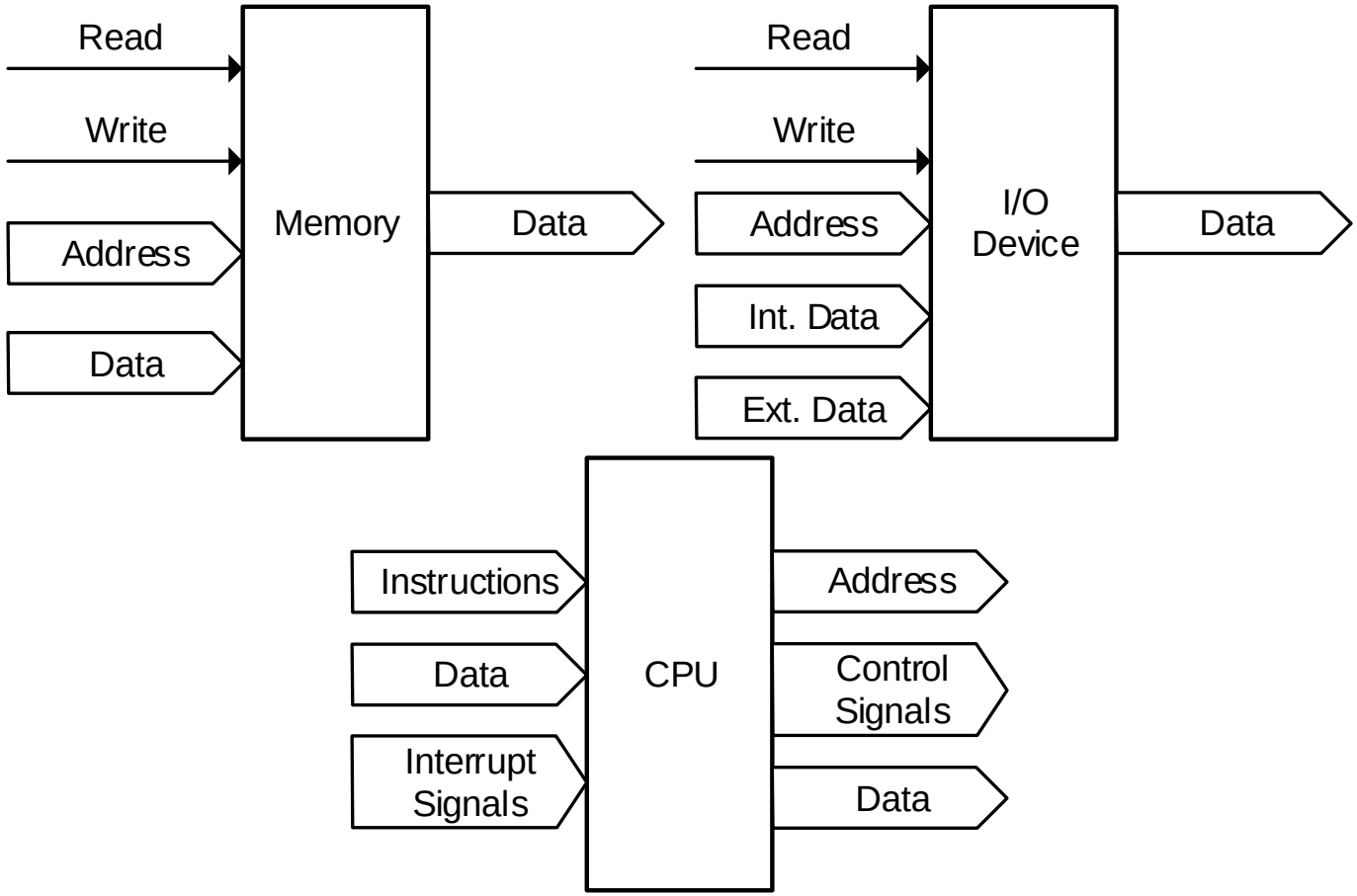
Mostrar no Gerenciador de Dispositivos do Windows o Endereço, IRQ, DMA dos dispositivos de E/S.

Nem sempre as interrupções são sinais de *hardware*. Pode ser *software*;
Os tipos de interrupção são:

1. **E/S** – geradas por um módulo/canal/controlador de E/S para sinalizar término de operação ou falhas;
2. ***Hardware*** – geradas por uma falha de hardware, como: erro de paridade de memória, falta de energia etc.
3. **Programa** – geradas devido à execução de uma instrução, como: *overflow* aritmético, **divisão por zero**, **falha de segmentação**, execução ilegal de instrução etc;
4. ***Timer*** – geradas por um temporizador dentro dos processos. SOs utilizam *timers* para realizar certas funções periodicamente.

Estrutura de interconexão

Antes de entender as interconexões, o que cada dispositivo envia e recebe?



Estrutura de interconexão

E o que cada dispositivo é ou faz?

- **Memória** – um conjunto de N palavras de mesmo comprimento endereçadas por sua posição, de 0 a $(N-1)$. As palavras podem ser lidas ou escritas;
- **Módulo/canal/controlador de E/S** – é semelhante à memória para a CPU. Pode controlar diversos dispositivos de E/S conectado a uma das M portas. É capaz de trocar dados e sinais de interrupção entre CPU e dispositivos de E/S;
- **CPU** – lê instruções e dados, escreve dados, emite sinais de controle para todo o sistema e recebe sinais de interrupção.

Estrutura de interconexão

Finalmente, como é o tráfego entre os dispositivos?

- **Memória** → **CPU** – a CPU lê uma palavra da memória (dados ou instruções);
- **CPU** → **Memória** – a CPU escreve uma palavra na memória;
- **E/S** → **CPU** – a CPU lê dados de um dispositivo de E/S utilizando um módulo/canal/controlador de E/S;
- **CPU** → **E/S** – a CPU grava dados em um dispositivo de E/S via controlador de E/S;
- **E/S** → **Memória** ou **Memória** → **E/S** – o controlador de E/S troca dados diretamente com a memória, sem a intervenção da CPU;

Interconexão de barramento

- **Barramento** (*bus*) conecta dois ou mais dispositivos e é um canal de transmissão **compartilhado**;
- Um sinal transmitido por um dispositivo está disponível para todos os outros ligados ao mesmo barramento;
- Se dois ou mais dispositivos transmitirem sinais ao mesmo tempo ocorrerá a **colisão**;
- Um barramento é formado por diversas linhas de transmissão;
- Cada linha transporta um bit. Várias linhas transportam diversos bits **em paralelo**;
- Um barramento que conecta os componentes principais do sistema é chamado de **barramento de sistema** (*system bus*).

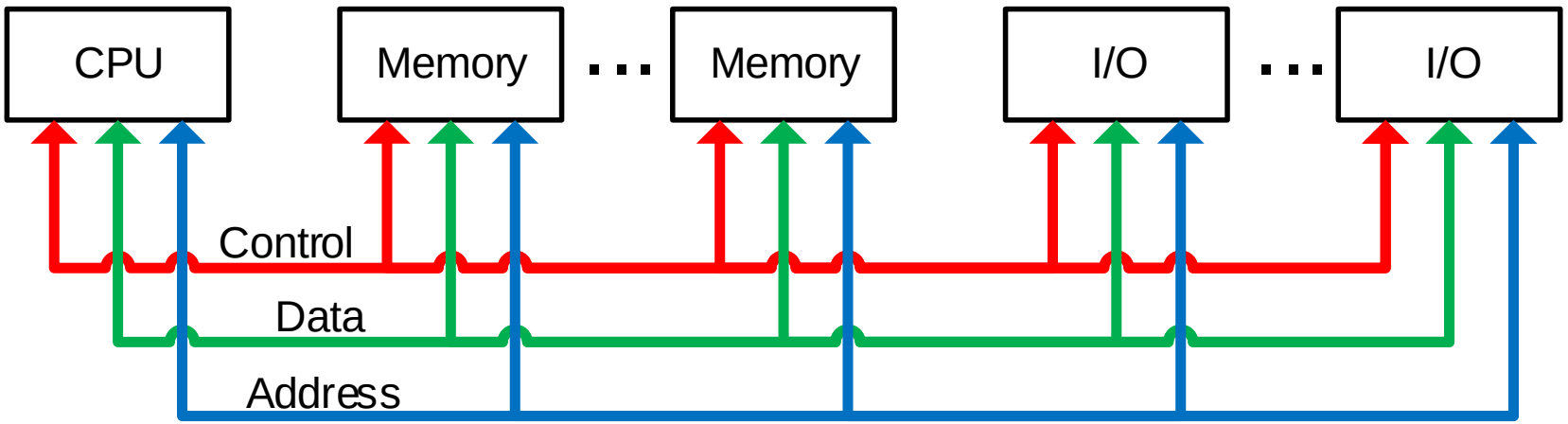
Interconexão de barramento

- Cada linha de transmissão possui uma função específica, entretanto, há três tipos:
  1. **Linhas de dados** – transfere dados entre componentes. Um barramento de 64 bits significa que tem 64 linhas de dados e 64 bits podem ser transferidos por ciclo de *clock*. É a **largura** do barramento. Bom exemplo: Intel 8086/8088;
  2. **Linhas de endereço** – indica a origem ou o destino dos dados. Em outras palavras, é o endereço do dado. Também pode ser usado para endereçar controlador de E/S e dispositivo de E/S usando partes da palavra de endereço. Exemplo: $10010110 \rightarrow$ controlador 2 e dispositivo 22;

Interconexão de barramento (cont'd)

3. **Linhas de controle** – transfere sinais de controle que podem ser:
   - ***Memory Read*** – dados de um end. mem. são colocados no *bus*;
   - ***Memory Write*** – dados no *bus* são gravados em um end. mem.;
   - ***I/O Read*** – dados de um disp. E/S são colocados no *bus*;
   - ***I/O Write*** – dados no *bus* são enviados para um disp. E/S;
   - ***Transfer* ACK** – indica que dados do *bus* foram aceitos ou que colocados nele;
   - ***Bus request*** – indica que um módulo precisa obter o controle do *bus*;
   - ***Bus grant*** – indica que um módulo recebeu o controle do *bus*;
   - ***Interrupt request*** – indica interrupção pendente;
   - ***Interrupt* ACK** – indica que uma interrupção foi aceita;
   - ***CLK*** – sincroniza as operações do *bus*;
   - ***Reset*** – inicializa todos os componentes.

Interconexão de barramento



Interconexão de barramento

Procedimento para um componente enviar dados para outro pelo *bus*:

1. Requisitar o controle do *bus*;
2. Esperar pela concessão do *bus*;
3. Transferir dados pelo *bus*.

Procedimento para um componente requisitar dados de outro pelo *bus*:

1. Requisitar o controle do *bus*;
2. Esperar pela concessão do *bus*;
3. Transferir o sinal de controle de requisição de dados pelo *bus*;
4. Esperar pelos dados serem transferidos pelo *bus* em outro momento.

Interconexão de barramento

- Quanto mais dispositivos conectados ao *bus*, maior a queda de desempenho;
- *Bus* (fisicamente) mais comprido para acomodar todos os dispositivos, gerando atrasos de propagação do sinal;
- *Bus* mais próximo da capacidade máxima de transferência de dados, podendo virar um gargalo;
- Até certo ponto, soluções podem ser aumentar a largura do *bus* ou aumentar o CLK;
- E ainda assim é insuficiente. Exemplos de dispositivos de E/S super exigentes:
  1. Placas de vídeo (GPUs);
  2. Placas de rede 40 GbE (Gigabit Ethernet) e 100 GbE;
  3. SSDs de última geração.
- Solução adotada pela indústria: **hierarquia de múltiplos barramentos**.

Figura: Estrutura de barramentos de computadores muito antigos

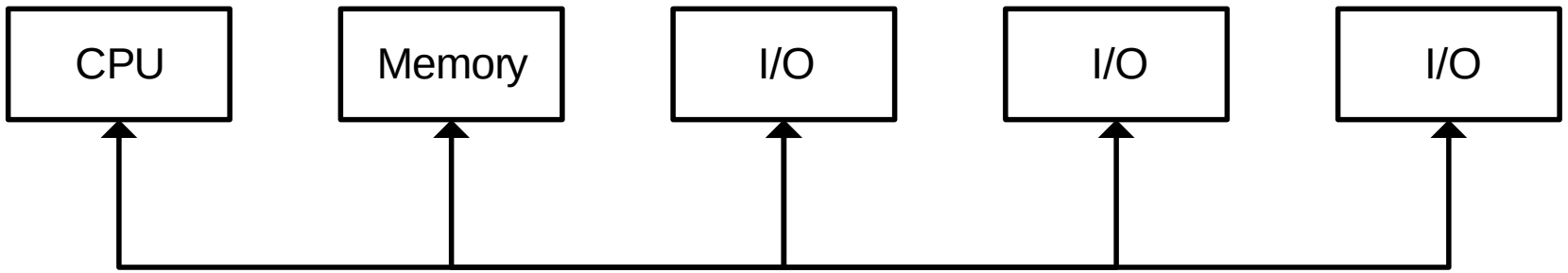


Figura: Estrutura de barramentos de PCs do final de 1990s

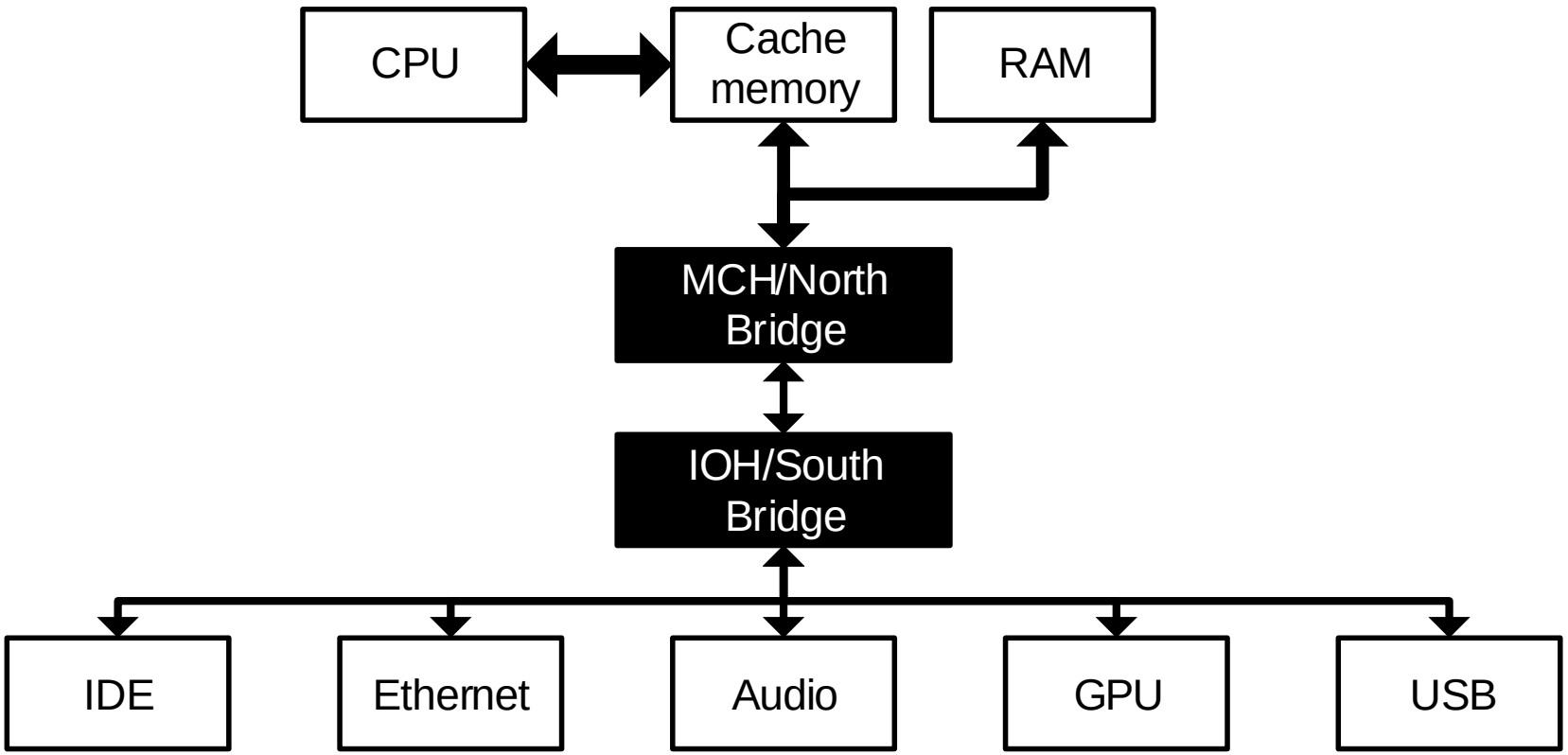


Figura: Estrutura de barramentos de PCs do final de 1990s até meados de 2000s

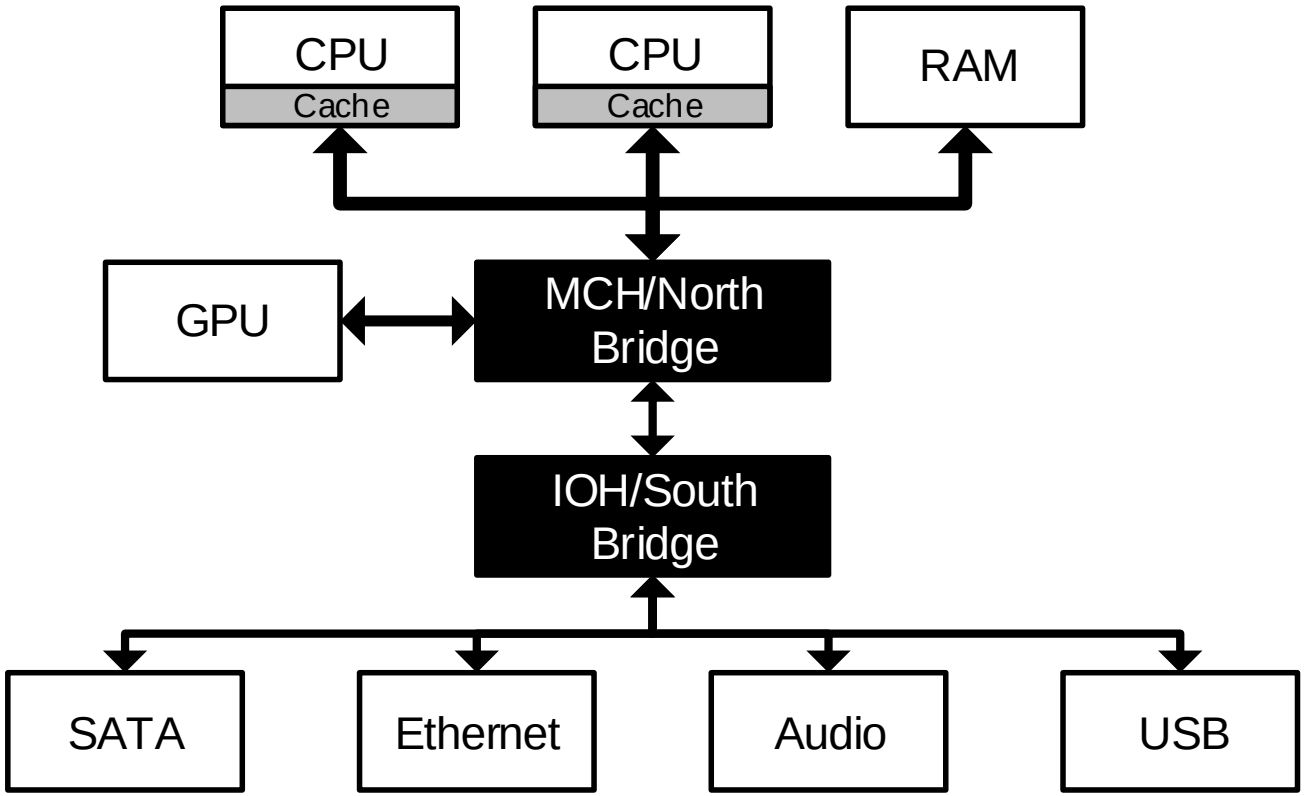


Figura: Estrutura de barramentos de PCs atuais

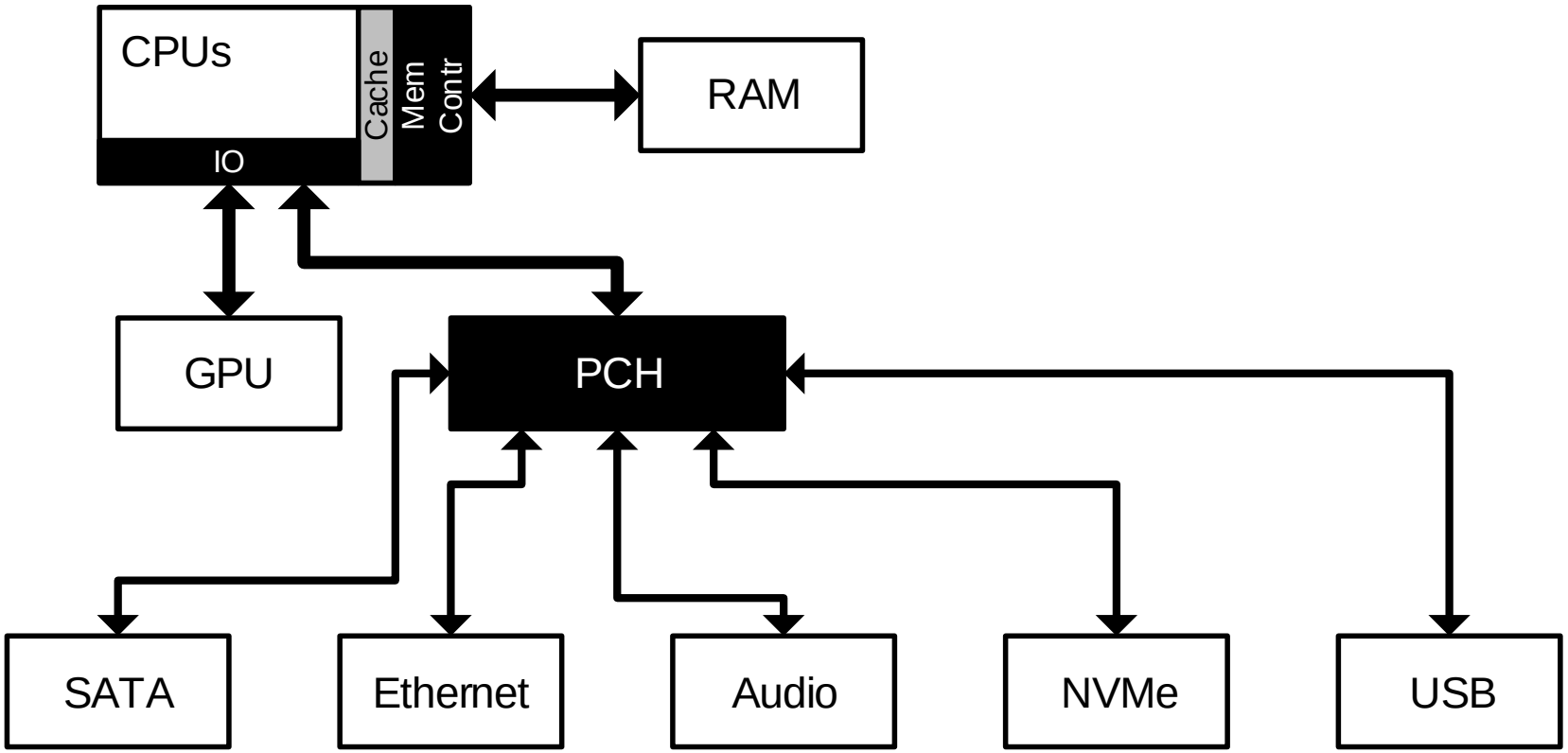
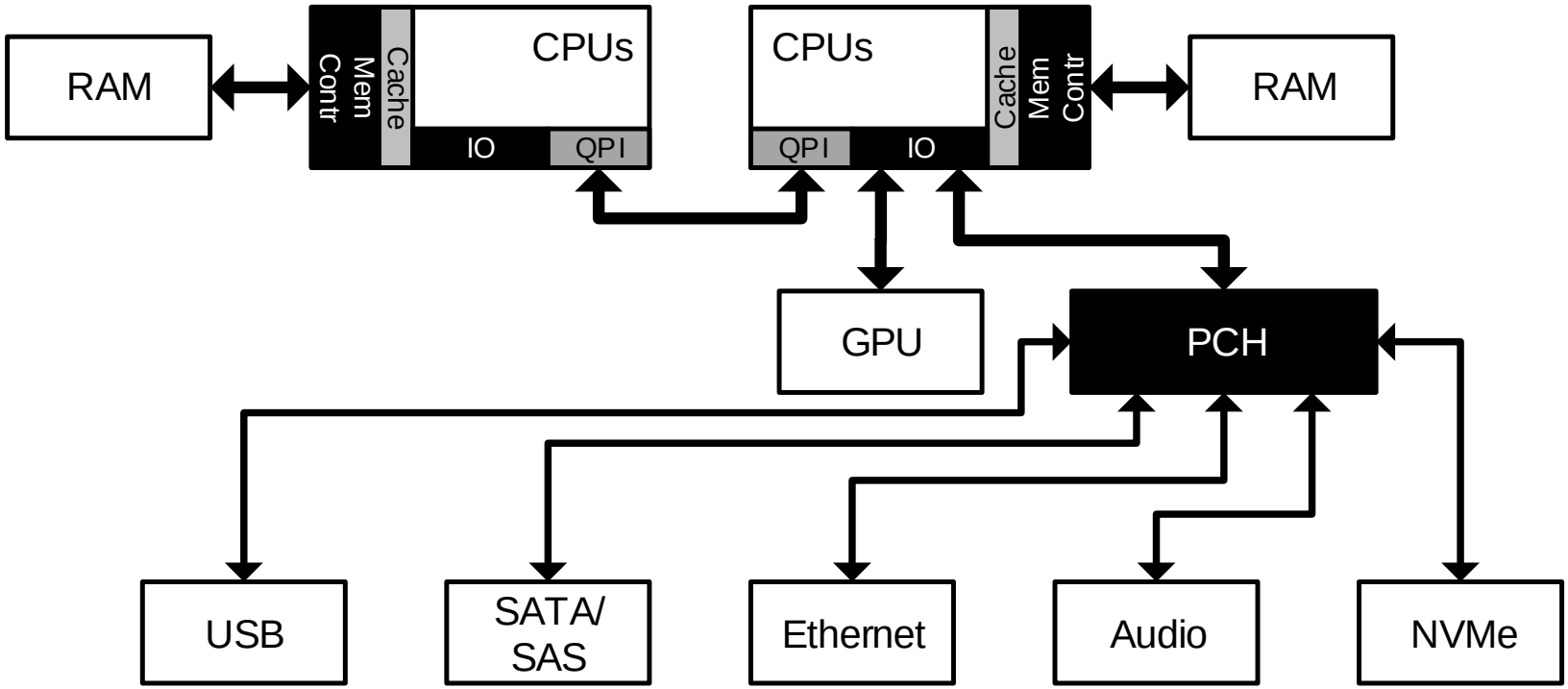


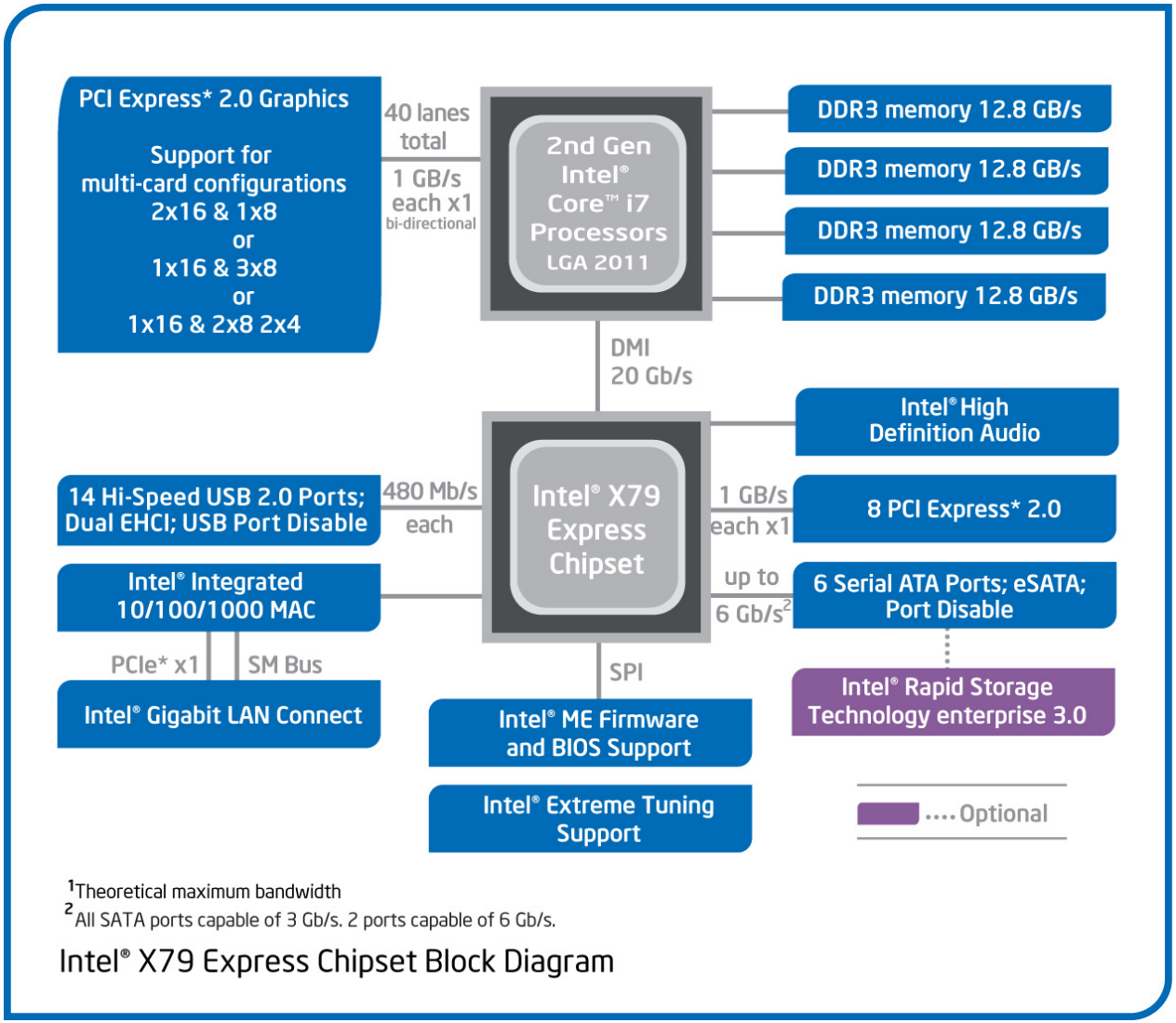
Figura: Estrutura de barramentos de workstations e servidores atuais



Interconexão de barramento

- Observa-se um isolamento da CPU em relação à memória (pela existência da memória cache) e dos dispositivos de E/S;
- Transferências E/S-memória e memória-E/S não afetam a CPU;
- Transferências CPU-memória e memória-CPU não afetam E/S;
- Isola inclusive dispositivos de E/S (mais próximos da CPU) rápidos dos dispositivos de E/S lentos (mais distantes da CPU).

Exemplo



[1] Theoretical maximum bandwidth
[2] All SATA ports capable of 3 Gb/s. 2 ports capable of 6 Gb/s.

Intel® X79 Express Chipset Block Diagram

Projeto de barramentos

Características básicas que regem o projeto de *buses*

- **Tipo** – pode ser dedicado ou multiplexado para funções ou determinados componentes do computador. Exemplos:
  - **Dedicação funcional**: linhas de endereço e de dados comumente são separadas, mas não é essencial;
  - **Multiplexação temporal**: as linhas de endereço e de dados são compartilhadas, dependendo se outro sinal de controle (*Address Valid*) estiver ativado ou não;
  - **Dedicação física**: Apenas alguns dispositivos utilizam um *bus*, em vez dele ser compartilhado com todos os outros.
  - Vantagens e desvantagens?

Projeto de barramentos (cont'd)

Características básicas que regem o projeto de *buses*

- **Método de arbitração** – apenas um dispositivo deve utilizar o *bus* por vez. Quem controla (ou arbitra) o acesso a ele?
  - **Central**: apenas um dispositivo – o controlador do *bus* – aloca/reserva o tempo de barramento para os outros dispositivos;
  - **Distribuída**: todos os dispositivos conectados ao *bus* controlam o acesso ao barramento de maneira compartilhada/cooperativa.
  - Vantagens e desvantagens?

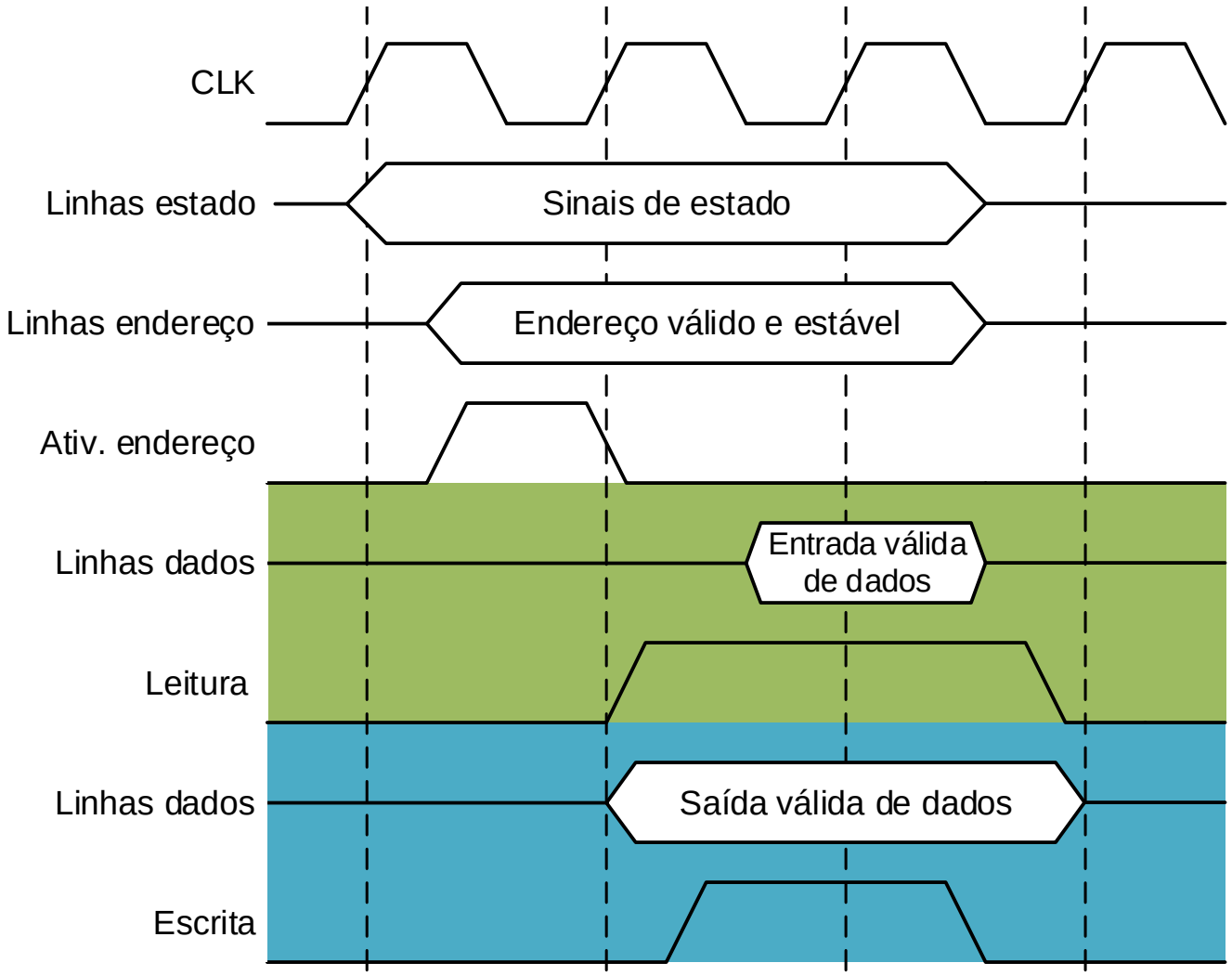
Projeto de barramentos (cont'd)

Características básicas que regem o projeto de *buses*

- **Temporização** – como os eventos são coordenados no *bus*?
  - **Síncrona**: todos os eventos são regidos pelo sinal CLK;
  - **Assíncrona**: a ocorrência dos eventos dependem dos eventos anteriores
  - Vantagens e desvantagens?

Projeto de barramentos (cont'd)

Figura: Sinais em um barramento síncrono



Projeto de barramentos (cont'd)

Características básicas que regem o projeto de *buses*

- **Largura do barramento** – quantos bits são transmitidos simultaneamente pelo *bus* (serial ou paralelo)? Vantagens e desvantagens?
- **Tipo de transferência de dados** – como os dados são transferidos pelo *bus*? A escrita (mestre-escravo) é multiplexada, enviando o endereço, depois os dados? Ou é simultaneamente? O *bus* permite operações *Read-Modify-Write* e *Read-After-Write*? Como são feitas? É necessário esperar pelo tempo de acesso? A leitura (escravo-mestre) é multiplexada ou não?

Exemplos de *buses*

Buses internos comuns:

- **ISA** - *Industry Standard Architecture*;
- **PCI** - *Peripheral Component Interconnect* 1.0, 2.0;
- **AGP** - *Accelerated Graphics Port* 1.0, 2.0, 3.0;
- **PCI-X** - *PCI Extended* 1.0, 2.0;
- **PCIe** - *PCI Express* 1.0, 2.0, 3.0 (x1, x4, x8, x16).

Buses externos comuns:

- **USB** - *Universal Serial Bus* 1.0, 2.0, 3.0;
- **Lightning**;
- **Thunderbolt** 1, 2, 3;
- **PATA/IDE** - Parallel ATA ou IDE;
- **SATA** - Serial ATA.

Visualizar fotos na pasta.

Exercícios

Capítulo abordado: 3